

GNU recutils

for version 1.5, 10 January 2012

GNU recutils Developers (bug-recutils@gnu.org)

This manual is for GNU recutils (version 1.5, 10 January 2012).

Copyright © 2009, 2010, 2011, 2012 Jose E. Marchesi

Copyright © 1994-2012 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

Table of Contents

| | | |
|-----------|---------------------------------------|-----------|
| 1 | Introduction | 1 |
| 2 | A Little Example | 2 |
| 3 | Fields and Records | 4 |
| 4 | Comments | 6 |
| 5 | Record Descriptors | 7 |
| 6 | %rec | 9 |
| 7 | %mandatory and %prohibit | 10 |
| 8 | %unique and %key | 11 |
| 9 | %doc | 12 |
| 10 | %typedef and %type | 13 |
| 10.1 | Declaring types | 13 |
| 10.2 | Types and fields | 14 |
| 10.3 | Escalar types | 14 |
| 10.4 | String types | 15 |
| 10.5 | Enumerated types | 15 |
| 10.6 | Time related types | 16 |
| 10.7 | Other types | 16 |
| 11 | %auto | 17 |
| 12 | %sort | 19 |
| 13 | %size | 20 |
| 14 | %confidential | 21 |
| 15 | Compound Field Names | 23 |

| | | |
|-----------|--|-----------|
| 16 | Date input formats | 25 |
| 16.1 | General date syntax | 25 |
| 16.2 | Calendar date items | 26 |
| 16.3 | Time of day items | 27 |
| 16.4 | Time zone items | 27 |
| 16.5 | Combined date and time of day items | 28 |
| 16.6 | Day of week items | 28 |
| 16.7 | Relative items in date strings | 28 |
| 16.8 | Pure numbers in date strings | 29 |
| 16.9 | Seconds since the Epoch | 29 |
| 16.10 | Specifying time zone rules | 30 |
| 16.11 | Authors of <code>parse_datetime</code> | 30 |
| 17 | Regular Expressions | 31 |
| 18 | Common Options | 32 |
| 18.1 | Selection Expressions | 32 |
| 18.1.1 | Operands | 32 |
| 18.1.1.1 | Numeric Literals | 32 |
| 18.1.1.2 | String Literals | 32 |
| 18.1.1.3 | Field Values | 32 |
| 18.1.1.4 | Parenthesized Expressions | 33 |
| 18.1.2 | Operators | 33 |
| 18.1.2.1 | Arithmetic Operators | 33 |
| 18.1.2.2 | Boolean Operators | 33 |
| 18.1.2.3 | Comparison Operators | 34 |
| 18.1.2.4 | Date Comparison Operators | 34 |
| 18.1.2.5 | Field Operators | 34 |
| 18.1.2.6 | String Operators | 34 |
| 18.1.2.7 | Conditional Operator | 34 |
| 18.1.3 | Evaluation of Selection Expressions | 34 |
| 18.2 | Field Expressions | 35 |
| 19 | recinf | 36 |
| 19.1 | recinf Invocation | 36 |
| 20 | recsel | 37 |
| 20.1 | recsel Invocation | 37 |
| 20.2 | recsel Examples | 38 |
| 20.3 | recsel Encryption | 38 |
| 21 | recins | 41 |
| 21.1 | recins Invocation | 41 |
| 21.2 | recins Examples | 42 |
| 21.3 | recins Encryption | 42 |
| 22 | recdel | 43 |
| 22.1 | recdel Invocation | 43 |
| 22.2 | recdel Examples | 43 |

| | | |
|-------------------|---|-----------|
| 23 | reset | 44 |
| 23.1 | reset Invocation | 44 |
| 23.2 | reset Examples | 45 |
| 24 | refix | 46 |
| 24.1 | refix Invocation | 46 |
| 24.2 | refix Examples | 46 |
| 25 | refmt | 48 |
| 25.1 | refmt Invocation | 48 |
| 25.2 | refmt Templates | 48 |
| 25.3 | refmt Examples | 49 |
| 26 | csv2rec | 50 |
| 26.1 | csv2rec Invocation | 50 |
| 26.2 | csv2rec Examples | 50 |
| 27 | rec2csv | 51 |
| 27.1 | rec2csv Invocation | 51 |
| 27.2 | rec2csv Conversion | 51 |
| 27.3 | rec2csv Examples | 52 |
| 28 | mdb2rec | 53 |
| 28.1 | mdb2rec Invocation | 53 |
| 28.2 | mdb2rec Examples | 53 |
| Appendix A | GNU Free Documentation License | 56 |

1 Introduction

GNU recutils is a set of tools and libraries to access human-editable, text-based databases called *recfiles*. The data is stored as a sequence of records, each record containing an arbitrary number of named fields. Advanced capabilities usually found in other data storage systems are supported: data types, data integrity (keys, mandatory fields, etc) as well as the ability of records to refer to other records (sort of foreign keys). Despite its simplicity, recfiles can be used to store medium-sized databases.

So, yet another data storage system? The mere existence of this package deserves an explanation. There is a rich set of already available free data storage systems, covering a broad range of requirements. Big systems having complex data storage requirements will probably make use of some full-fledged relational system such as MySQL or PostgreSQL. Less demanding applications, or applications with special deployment requirements, may find it more convenient to use a simpler system such as SQLite, where the data is stored in a single binary file. XML files are often used to store configuration settings for programs, and to encode data to be transmitted through networks.

So it looks like all the needs are covered by the existing solutions. . . but consider the following characteristics of the data storage systems mentioned in the previous paragraph:

- The stored data is not directly readable by humans.
- The stored data is definitely not directly writable by humans.
- They are program dependent.
- They are not easily managed by version control systems.

Regarding the first point (human readability), while it is clearly true for the binary files, some may argue XML files are indeed human readable. . . well. . . `<bar><foo tag="val">try</foo> to read <p>this</p></bar>`. YAML¹ is an example of a hierarchical data storage format which is much more readable than XML. The problem with YAML is that it was designed as a “data serialization language” and thus to map the data constructs usually found in programming languages. That makes it too complex for the simple task of storing plain lists of items.

Recfiles are human-readable, human-writable and still they are easy to parse and to manipulate automatically. Obviously they are not suitable for any task (for example, it can be difficult to manage hierarchies in recfiles) and performance is somewhat sacrificed in favor of readability. But they are quite handy to store small to medium simple databases.

The GNU recutils suite comprises:

- This texinfo manual, describing the Rec format and the accompanying software.
- A C library (`librec`) that provides a rich set of functions to manipulate rec data.
- A set of C utilities (`recinf`, `recsel`, `recins`, `recdel`, `recset` and `recfix`) that can be used in shell scripts and in the command line to operate on rec files.
- An emacs mode, `rec-mode`.

¹ Yet Another Markup Language

2 A Little Example

Everyone loves to grow a nice book collection at home. Unfortunately, in most cases the management of our private books gets uncontrolled: some books get lost, some of them may be loaned to some friend, there are some duplicated (or even triplicated!) titles because we forgot about the existence of the previous copy, and many more details.

In order to improve the management of our little book collection we could make use of a complex data storage system such as a relational database. The problem with that approach, as explained in the previous section, is that the tool is too complicated for the simple task: we do not need the full power of a relational database system to maintain a simple collection of books.

With GNU recutils it is possible to maintain such a little database in a text file. Let's call it 'books.rec'. The following table resumes the information items that we want to store for each title, along with some common-sense restrictions.

- Every book has a title, even if it is "No Title".
- A book can have several titles.
- A book can have more than one author.
- Sometimes the author is not known.
- Sometimes we don't care about who the author of a book is.
- We usually store our books at home.
- We use to loan books to friends.
- On occasions we loose track to the physical location of a book. Did we loan it to anyone? Was it lost in the last move? Is it in some hidden place at home?

The contents of the rec file follows:

```
# -*- mode: rec -*-

%rec: Book
%mandatory: Title
%type: Location enum loaned home unknown
%doc:
+ A book in my personal collection.

Title: GNU Emacs Manual
Author: Richard M. Stallman
Publisher: FSF
Location: home

Title: The Colour of Magic
Author: Terry Pratchett
Location: loaned

Title: Mio Cid
Author: Anonymous
Location: home

Title: chapters.gnu.org administration guide
Author: Nacho Gonzalez
Author: Jose E. Marchesi
Location: unknown
```

```
Title: Yeelong User Manual
Location: home
```

```
# End of books.rec
```

Simple. The file contains a set of records separated by blank lines. Each record is composed by a set of fields with a name and a value.

The GNU recutils can then be used to access the contents of the file. For example, we could get a list of the names of loaned books invoking `recsel` in the following way:

```
$ recsel -e "Location = 'loaned'" -P Title books.rec
The Colour of Magic
```

3 Fields and Records

A *field* is the written form of an association between a label and a value. For example, if we wanted to associate the label `Name` with the value `Ada Lovelace` we would write:

```
Name: Ada Lovelace
```

The separator between the field name and the field value is a colon followed by a blank character (space and tabs, but not newlines). The name of the field shall begin in the first column of the line.

A *field name* is a sequence of alphanumeric characters plus dashes (-) and underscores (_), starting with a letter or the character %. The regular expression denoting a field name is:

```
[a-zA-Z%][a-zA-Z0-9_-]+
```

Field names are case-sensitive. `Foo` and `foo` are considered different field names.

The following list contains valid field names (the final colon is not part of the names):

```
Foo:
foo:
A23:
ab1:
A_Field:
```

The *value of a field* is a sequence of characters terminated by a single newline character (`\n`).

Sometimes a value is too long to fit in the usual width of terminals and screens. In that case, depending on the specific tool used to access the file, the readability of the data would not be that good. It is therefore possible to physically split a logical line by escaping a newline with a backslash character, as in:

```
LongLine: This is a quite long value \
composed by a unique logical line \
split in several physical lines.
```

The sequence `\n` (newline) + (PLUS) and an optional `_` (SPACE) is interpreted as a newline when found in a field value. For example, the C string `"bar1\nbar2\n bar3"` would be encoded in the following way in a field value:

```
Foo: bar1
+ bar2
+ bar3
```

A *record* is a group of one or more fields written one after the other:

```
Name1: Value1
Name2: Value2
Name2: Value3
```

It is possible for several fields in a record to share the same name or/and the field value. The following is a valid record containing three fields:

```
Name: John Smith
Email: john.smith@foomail.com
Email: john@smith.name
```

The *size of a record* is defined as the number of fields that it contains. There is no such like an empty record, so the minimum size for a record is 1. The maximum number of fields for a record is only limited by the available physical resources. The size of the previous record is 3.

Records are separated by one or more blank lines. For instance, the following example shows a file named `'personalities.rec'` featuring three records:

Name: Ada Lovelace

Age: 36

Name: Peter the Great

Age: 53

Name: Matusalem

Age: 969

4 Comments

Any line having an # (ASCII 0x23) character in the first column is a comment line.

Comment lines are quite useful to insert additional information that is not part of the database but useful otherwise.

It is also quite convenient to comment-out information from the recfile without having to remove it in a definitive way: you may want to recover the data into the database later! Comment lines can be used to comment-out both full registers and single fields:

```
Name: Jose E. Marchesi
# Occupation: Software Engineer
# Severe lack of brain capacity
# Fired on 02/01/2009 (without compensation)
Occupation: Unoccupied
```

Comments are also useful for headers, footers, comment blocks and all kind of markers:

```
# -*- mode: rec -*-
#
# TODO
#
# This file contains the Bugs database of GNU recutils.
#
# Blah blah...

...

# End of TODO
```

5 Record Descriptors

Certain properties of a set of records can be specified by preceding them with a *record descriptor*. A record descriptor is itself a record, and uses fields with some predefined names to store the properties. The most basic property that can be specified for a set of records is their *type*. The special field name `%rec` is used for that purpose:

```
%rec: Entry

Id: 1
Name: Entry 1

Id: 2
Name: Entry 2
```

The records following the descriptors are then identified as having its type. So in the example above we would say there are two records of type “Entry”.

The effect of a record descriptor ends when another descriptor is found in the stream of records. This allows to store different kind of records in the same database. For example, consider you have to maintain a depot. You will need to keep records of both the current stockage and the movements.

The following example shows the usage of two record descriptors to store both kind of records: articles and movements.

```
%rec: Article

Id: 1
Title: Article 1

Id: 2
Title: Article 2

%rec: Movement

Id: 1
Type: sell
Date: 20 April 2011

Id: 2
Type: acquisition
Date: 21 April 2011
```

Besides determining the type of the records that follows in the stream, record descriptors can be used to describe other properties of those records. That can be done by using the so-called *special fields*, having special names from a predefined set. Consider for example the following database, where the descriptor is used to specify a primary key and a mandatory field:

```
%rec: Item
%key: Id
%mandatory: Title

Id: 10
Title: Notebook (big)

Id: 11
```

Title: Fountain Pen

Note that the names of special fields always start with the character `%`. Also note that it is also possible to use non-special fields in a record descriptor, but such fields will have no effect on the described record set.

What follows is an exhaustive list of the supported special fields. They are discussed in deep in the following sections.

`%rec` Used to name the type of a set of records and to mark record descriptors.

`%mandatory`

`%prohibit`

Those special fields are used to control which fields can and cannot be included in records.

`%unique`

`%key` Used to specify unique fields and primary keys.

`%auto` Used to automatically generate certain fields when creating new records, like auto counter and time-stamps.

`%doc` Used to describe the contents of a record set.

`%typedef`

`%type` Used to define named types and to associated types with fields.

`%sort` Used to keep your records sorted by some given field.

`%size` Used to control the dimensions of a record set.

`%confidential`

Used to store confidential information.

6 %rec

The %rec special field is used for two main purposes: to identify a record as a record descriptor, and to provide a name for the described record set. The synopsis of the usage of the field is the following:

```
%rec: type [url_or_file]
```

type is the name of the kind of records described by the descriptor. It is mandatory to specify it, and it follows the same lexical conventions used by field names. See [Chapter 3 \[Fields and Records\]](#), page 4. There is a non-enforced convention to use singular nouns, because the name makes reference to the type of a single entity, even if it applies to all the records contained in the record set. For example, the following record set contains transactions, and the type specified in the record descriptor is **Transaction**.

```
%rec: Transaction
```

```
Id: 10
Title: House rent
```

```
Id: 11
Title: Loan
```

Only one %rec field shall appear in a record descriptor. If there are more it is considered an integrity violation. It is highly recommended (but not enforced) to place this field in the first position of the record descriptor.

Sometimes it is convenient to store records of the same type in different files. The duplication of record descriptors in this case would surely lead to consistency problems. A possible solution would be to keep the record descriptor in a separated file and then include it in any operation by using pipes. For example:

```
$ cat descriptor.rec data.rec | recsel ...
```

For those cases it is more convenient to use a *external descriptor*. External descriptors can be built appending a file path to the %rec field value, like:

```
%rec: FSD_Entry /path/to/file.rec
```

The previous example indicates that a record descriptor describing the FSD_Entry records shall be read from the file '/path/to/file.rec'. A record descriptor for FSD_Entry may not exist in the external file. Both relative and absolute paths can be specified there.

URLs can be used as sources for external descriptors as well. In that case we talk about *remote descriptors*. For example:

```
%rec: Department http://www.myorg.com/Org.rec
```

The URL shall point to a text file containing rec data. If there is a record descriptor in the remote file documenting the **Department** type, it will be used.

Note that the local record descriptor can provide additional fields to “expand” the record type. For example:

```
%rec: FSD_Entry http://www.jemarch.net/downloads/FSD.rec
%mandatory: Rating
```

The record descriptor above is including the contents of the FSD_Entry record descriptor from the URL, and adding them to the local record descriptor, that in this case contains just the %mandatory field.

If you are using the GNU recutils to process your recfiles, any URL schema supported by libcurl will work.

7 %mandatory and %prohibit

Those special field names are used to restrict the fields that can appear in the records stored in a database. Their usage is:

```
%mandatory: field1 field2 ... fieldN
%prohibit: field1 field2 ... fieldN
```

In both cases the list of field names are separated by one or more blank characters.

The fields listed in some %mandatory entry will be considered mandatory; i.e., at least one field with this name shall be present in any record of this kind. Records violating this restriction will be considered invalid and a checking tool will report the situation as a data integrity failure.

Consider for example an “addressbook” database where each record stores the information associated with a contact. The records will be heterogeneous, in the sense they won’t feature exactly the same fields: the contact of an internet shop will probably have an `Url` field, while the entry for our grandmother probably won’t. We still want to make sure that every entry has at a field: the name of the contact. In that case we could use %mandatory as follows:

```
%rec: Contact
%mandatory: Name
```

```
Name: Granny
Phone: +12 23456677
```

```
Name: Yoyodyne Corp.
Email: sales@yoyod.com
Phone: +98 43434433
```

Similarly, the fields listed in some %prohibit entry will be considered forbidden; i.e., no field with this name shall be present in any record of this kind. Again, records violating this restriction will be considered invalid.

This can be useful when some field name is reserved for some future use. For example, if we want to make sure no `Telex` will be even used in our contacts database, we would change the record descriptor as follows:

```
%rec: Contact
%mandatory: Name
%prohibit: Telex
```

Several %mandatory and/or %prohibit fields can appear in the same record descriptor. The set of mandatory or prohibited fields is the union of all the entries. For example, in the following database both `Id` and `id` are prohibited:

```
%rec: Entry
%prohibit: Id
%prohibit: id
```

8 %unique and %key

These special field names are used to avoid several instances of the same field in a record, and to implement keys in record sets. Their usage is:

```
%unique: field1 field2 ... fieldN
%key: field
```

The list of field names are separated by one or more blank characters.

The %unique special field allows to declare fields as unique, meaning there cannot exist more than one field with the same name in a single record.

For example, an entry in an addressbook database could contain an **Age** field. It does not make sense for a single person to be of several ages, so that field could be declared as “unique” in the corresponding record descriptor as follows:

```
%rec: Contact
%mandatory: Name
%unique: Age
```

Several %unique fields can appear in the same record descriptor. The set of unique fields is the union of all the entries.

%key makes the referred field the primary key of the record set. Its effect is that any field with that name must be both unique and mandatory, and additionally the values of those fields shall be unique in the context of the record set. This closely corresponds to the notion of “primary key” usually implemented in the relational systems.

Consider for example a database of items in a stockage. Each item is identified by a numerical **Id** field. No item may have more than one **Id**, and no items may exist without an associated **Id**. Additionally, no two items may share the same **Id**. This common situation can be implementing by declaring **Id** as the key in the record descriptor:

```
%rec: Item
%key: Id
%mandatory: Title
```

```
Id: 1
Title: Box
```

```
Id: 2
Title: Sticker big
```

It would not make sense to have several primary keys in a record set, and thus it is not allowed to have several %key fields in the same record descriptor. That situation is considered a data integrity violation and will be reported by a checking tool.

9 %doc

This field contains documentation about the record.

It is like a comment, but this field can be managed in a programmatic way easier.

10 %typedef and %type

Field values are, by default, unrestricted text strings. However, it is often useful to impose some restrictions on the values of certain fields. For example, consider the following record:

```
Id: 111
Name: Jose E. Marchesi
Age: 30
MaritalStatus: single
Phone: +49 666 666 66
```

Where `Id` is a numeric identifier for a person. `Name` will never use several lines. `Age` will typically be in the range 0..120, and there are only a few valid values for `MaritalStatus`: single, married and widow. Phones may be restricted to some standard format as well to be valid. All those restrictions (and many others) can be enforced by using *field types*.

There are two kind of field types: anonymous and named. Those are described in the following subsections.

10.1 Declaring types

A type can be declared in a record descriptor by using the `%typedef` special field. The syntax is:

```
%typedef: type_name type_description
```

Where *type_name* is the name of the new type, and *type_description* a description which varies depending of the kind of type. For example, this is how a type `Age_t` could be defined as numbers in the range 0..120:

```
%typedef: Age_t range 0 120
```

Type names are identifiers having the following syntax:

```
[a-zA-Z][a-zA-Z0-9_-]*
```

Even though any identifier with that syntax could be used for types, it is a good idea to consistently follow some convention to help distinguishing type names from field names. For example, the `_t` suffix could be used for types.

A type can be declared to be a synonym of another type. The syntax is:

```
%typedef: type_name other_type_name
```

Where *type_name* is declared to be a synonym of *other_type_name*. This is useful to avoid duplicated type descriptions. For example, consider the following example:

```
%typedef: Id_t          int
%typedef: Item_t        Id_t
%typedef: Transaction_t Id_t
```

Both `Item_t` and `Transaction_t` are synonyms for the type `Id_t`. They are both numeric identifiers.

The order of the `%typedef` fields is not relevant. In particular, a type definition can reference other type that is defined below. The previous example could have been written as:

```
%typedef: Item_t        Id_t
%typedef: Transaction_t Id_t
%typedef: Id_t          int
```

Integrity checks will complain if undefined types are referenced, and if there are loops (direct or indirect) in type declarations. For example, the following set of declarations contains a loop and are thus invalid:

```
%typedef: A_t B_t
%typedef: B_t C_t
%typedef: C_t A_t
```

The scope of a type is the record descriptor where it is defined.

10.2 Types and fields

Fields can be declared to have a given type by using the %type special field in a record descriptor. The synopsis is:

```
%type: field_list type_name_or_description
```

Where *field_list* is a list of field names separated by commas. *type_name_or_description* can be either a type name declared with %typedef, or a type description. Type names are useful when several fields are declared to be of the same type:

```
%typedef: Id_t    int
%type:      Id      Id_t
%type:      Product Id_t
```

Anonymous types can be used by writing a type description instead of a type name. They help to avoid superfluous type declarations. A record containing a single Id field, for example, can be defined without having to use a %typedef in the following way:

```
%rec: Task
%type: Id int
```

10.3 Escalar types

The rec format supports the declaration of fields of the following escalar types: integer numbers, ranges and reals.

Signed *integers* are supported by using the *integer* declaration:

```
%typedef: Id_t int
```

Given that declaration, Fields of type Id_t must contain integers, that may be negative. Hexadecimal values can be written using the 0x prefix, and octal values use an extra 0. Valid examples are:

```
%type: Id Id_t

Id: 100
Id: -23
Id: -0xFF
Id: 020
```

Sometimes it is desirable to reduce the *range* of integers allowed in a field. That can be achieved by using a range type declaration:

```
%typedef: Percentage_t range 0 100
```

Note that it is possible to omit the minimum index in ranges. In that case it is implicitly zero:

```
%typedef: Percentage_t range 100
```

Hexadecimal and octal numbers can be used to specify the limits in a range. This helps to define scalar types whose natural base is not ten, like for example:

```
%typedef: Address_t range 0x0000 0xFFFF
%typedef: Perms_t range 755
```

Real fields can be declared with the *real* type specifier. A wide range of real numbers can be represented this way, only limited by the underlying floating point representation. For example:

```
%typedef: Longitude_t real
```

Examples of fields of type real:

```
%rec: Rectangle
%typedef: Longitude_t real
%type: Width Longitude_t
%type: Height Longitude_t
```

```
Width: 25.01
Height: 10
```

10.4 String types

The `line` field type specifier can be used to restrict the value of a field to a single line, i.e. no newline characters are allowed. For example, a type for proper names could be declared as:

```
%typedef: Name_t line
```

Examples of fields of type line:

```
Name: Mr. Foo Bar
Name: Mss. Bar Baz
Name: This is
+ invalid
```

Sometimes it is the maximum size of the field value that shall be restricted. The `size` field type specifier can be used to define the maximum number of characters a field value can have. For example, if we were collecting input that will get written in a paper-based forms system allowing up to 25 characters width entries, we could declare the entries as:

```
%typedef: Address_t size 25
```

Note that hexadecimal and octal integer constants can also be used to specify field sizes:

```
%typedef: Address_t size 0x18
```

Arbitrary restrictions can be defined by using regular expressions. The `regexp` field type specifier introduces an ERE (extended regular expression) that will be matched against fields having that name. The synopsis is:

```
%typedef: type_name regexp /re/
```

For example, consider the `Id_t` type designed to represent the encoding of the identifier of ID cards in some country:

```
%typedef: Id_t regexp /[0-9]{9}[a-zA-Z]/
```

Examples of fields of type `Id_t` are:

```
IDCard: 123456789Z
IDCard: invalid id card
```

Note that the slashes delimiting the RE can be replaced with any other character that is not itself used as part of the regexp. That is useful in some cases such as:

```
%typedef: Path_t regexp |(/[~/]?)+|
```

The regexp flavor supported in recfiles are the POSIX EREs plus several GNU extensions. See [Chapter 17 \[Regular Expressions\]](#), page 31.

10.5 Enumerated types

Fields of this type contain symbols taken from an enumeration.

The type is described by writing the sequence of symbols conforming the enumeration. Enumeration symbols are strings described by the following regexp:

```
[a-zA-Z0-9][a-zA-Z0-9_-]*
```

The symbols are separated by blank characters (including newlines). For example:

```
%typedef: Status_t enum NEW STARTED DONE CLOSED
%typedef: Day_t enum Monday Tuesday Wednesday Thursday Friday
+
+ Saturday Sunday
```

It is possible to insert comments when describing an enum type. The comments are delimited by parenthesis pairs. The contents of the comments can be any character but parenthesis. For example:

```
%typedef: TaskStatus_t enum
+ NEW (The task was just created)
+ IN_PROGRESS (Task started)
+ CLOSED (Task closed)
```

Boolean fields can be seen as special enumerations holding the binary values.

```
%typedef: Yesno_t bool
```

The literals allowed in boolean fields are *yes/no*, *0/1* and *true/false*. Examples are:

```
SwitchedOn: 1
SwitchedOn: yes
SwitchedOn: false
```

10.6 Time related types

The *Date* field type specifier can be used to declare dates and times. The synopsis is:

```
%typedef: type_name date
```

The several date/time syntax supported by librec is provided by the `date` module from gnulib. See [Chapter 16 \[Date input formats\]](#), page 25.

10.7 Other types

The *Email* field type specifier is used to declare electronic addresses such as emails. The synopsis is:

```
%typedef: Email_t email
```

Sometimes it is useful to make fields to store field names. For that purpose the *Field* field type specifier is supported. The synopsis is:

```
%typedef: Field_t field
```

11 %auto

This special field allows to include auto-generated fields in our databases. Its usage is:

```
%auto: field1 field2 ... fieldN
```

The list of field names are separated by one or more blank characters. There can be several %auto fields in the same record descriptor, the effective list of auto-generated fields being the union of all the entries.

Auto generated fields are a very useful facility usually provided by database implementations. Consider for example a list of articles in stock in a toys store:

```
%rec: Item
%key: Description

Description: 2cm metal soldier WWII
Amount: 2111

Description: Flying Helicopter Indoor Maxi
Amount: 8

...
```

It would be natural to identify the items by its description, but it is also error prone: was it “Flying Helicopter Indoor Maxi” or “Flying Helicopter Maxi Indoor”? Was “Helicopter” in lower case or upper case?

It is quite common in databases to use some kind of numeric “Id” to uniquely identify items like those ones. That is because numbers are easy to operate with, and to increase. So we could add a new numeric Id field and use it as the primary key:

```
%rec: Item
%key: Id
%mandatory: Description

Id: 0
Description: 2cm metal soldier WWII
Amount: 2111

Id: 1
Description: Flying Helicopter Indoor Maxi
Amount: 8

...
```

A problem of this approach is that we must be careful to not assign already used ids when we introduce more articles in the database. Other than its uniqueness, it is not important which number is associated with which article.

To ease the management of those Ids database systems use to provide a facility called “auto-counters”. Auto-counters can be implemented in recfiles using the %auto directive in the record descriptor:

```
%rec: Item
%key: Id
%type: Id int
%mandatory: Description
%auto: Id
```

```
Id: 0
Description: 2cm metal soldier WWII
Amount: 2111
```

Next time a new item is introduced in the database, the conforming application will note the %auto, and will create a new Id field for the new record with the bigger unused integer available. In this example, the new record will have an Id of 1. The application can still provide an explicit Id for the new record. In that case the field is not generated automatically.

The concrete effect of the %auto directive depends on the type of the affected field:

- If it is an **integer** or a **range** then the biggest non-used number in the record set will be generated.
- If it is a **date** then a field with the current date will be generated.
- If no explicit type is defined for an auto generated field it is assumed to be an integer.
- Auto generated fields with other types are not allowed.

Auto generated dates can be used to implement automatic timestamps. Consider for example a “Transfer” record set registering bank transfers. We want to save a timestamp every time a transfer is done, so we include an %auto for the date:

```
%rec: Transfer
%key: Id
%type: Id int
%type: Date date
%auto: Id Date
```

Note that the auto fields are generated at the beginning of the new records, in the same order they are found in the %auto directives.

12 %sort

This special field allows to set a sorting criteria for the records contained in a record set. Its usage is:

```
%sort: field_name
```

Meaning that the desired order for the records will be determined by the contents of the fields named `field_name`. The sorting is always done in ascending order, and there may be records not featuring fields named `field_name`, i.e. it is not mandatory for the sorting field to be mandatory :D.

It is an error to have more than one `%sort` field in the same record descriptor, as only one field can be used as sorting criteria.

Consider for example that we want to keep the records in our stockage system ordered by entry date. We could achieve that by using the following record descriptor in the database:

```
%rec: Item
%type: Date date
%sort: Date

Id: 1
Title: Staplers
Date: 10 February 2011

Id: 2
Title: Ruler Pack 20
Date: 2 March 2009

...
```

As you can see in the example above, the fact we use `%sort` in a database does not mean that the database will be always physically ordered. Unsorted record sets are not considered a data integrity problem, and thus the diagnosis tools must not declare a recfile as invalid because of this. Some tool may provide a way to physically order the fields in the file. That is certainly the case of the GNU recutils, where `recfix` can be used for that purpose. See [Chapter 24 \[recfix\]](#), page 46.

On the other hand any program listing, presenting or processing data extracted from the recfile must honour the `%sort` entry. For example, when using the following `recsel` program in the database above we would get the output sorted by date:

```
$ recsel stockage.rec
Id: 2
Title: Ruler Pack 20
Date: 2 March 2009

Id: 1
Title: Staplers
Date: 10 February 2011
```

The sorting of the selected field depends on its type:

- Numeric fields (integers, ranges, reals) are numerically ordered.
- Boolean fields are ordered considering that “false” values come first.
- Dates are ordered as expected.
- Any other kind of field is ordered using a lexicographic order.

13 %size

This special field is used to define constraints in the number of records stored in a record set. Its usage is:

```
%size: [relational_operator] number
```

If no operator is specified then *number* is interpreted as the exact number of records of this type. The number can be any integer literal, including hexadecimal and octal constants. For example:

```
%rec: Day
%size: 7
%type: Name enum
+ Monday Tuesday Wednesday Thursday Friday
+ Saturday Sunday
%doc: There should be 7 days.
```

The optional *relational_operator* shall be one of <, <=, > and >=. For example:

```
%rec: Item
%key: Id
%size: <= 100
%doc: We have at much 100 different articles.
```

It is valid to specify a size of 0, meaning that no records of this type shall exist in the file.

This field shall appear only once in a record descriptor.

14 %confidential

This special field allows to declare a set of fields as *confidential*, meaning they contain sensible information such as passwords or secrets. Its usage is:

```
%confidential: field1 field2 ... fieldN
```

The list of field names are separated by one or more blank characters. There can be several %confidential fields in the same record descriptor, the effective list of confidential fields being the union of all the entries.

Declaring a field as confidential indicates that its contents must not be stored in plain text, but encrypted with a password-based mechanism. When the information is retrieved from the database the confidential fields are unencrypted if the correct password is provided. Likewise, when information is inserted in the database the confidential fields are encrypted with some given password.

For example, consider a database of users of some service. For each user we want to store a name, a login name, an email address and a password. All this information is public with the obvious exception of the password. Thus we declare the Password field as confidential in the corresponding record descriptor:

```
%rec: Account
%type: Name line
%type: Login line
%type: Email email
%confidential: Password
```

The rec format does not impose the usage of a specific encryption algorithm, but requires that:

- The algorithm must be password-based.
- The value of any encrypted field shall begin with the string `encrypted-` followed by the encrypted data.
- The encrypted data must be encoded in some ASCII encoding such as base64.

The above rules assure that it is possible to determine whether a given field is encrypted. For example, the following is an excerpt from the account database described above. It contains an entry with the password encrypted and another with the password unencrypted:

```
Name: Mr. Foo
Login: foo
Email: foo@foo.com
Password: encrypted-AAABBBCCDDDEEEFFF

Name: Mr. Bar
Login: bar
Email: bar@bar.com
Password: secret
```

Unencrypted confidential fields are considered a data integrity error, and utilities like `recfix` will report it. The same utility can be used to “fix” the database by massively encrypting any unencrypted field.

Nothing prevents the usage of several passwords in the same database. This allows the establishment of several level of securities or security profiles. For example, we may want to store different passwords for different online services:

```
%rec: Account
%confidential: WebPassword ShellPassword
```

We could then encrypt WebPassword entries using a password shared among all the webmasters, and the ShellPassword entries with a more restricted password available only to the administrator of the machine.

The GNU recutils fully support encrypted fields. See the documentation for `recsel`, `recins` and `recfix` for details on how to operate on files containing confidential fields.

15 Compound Field Names

Please note that the features described in this chapter are very likely to change in the near future. It is thus recommended to not rely on them in your own databases.

It is possible to make a reference to a record (or set of records) of a certain type by using a *compound field name*. A compound field is composed by three components separated by double colon characters (:):

```
TYPE:FIELD[:ROLE]:
```

The first component is a record type, usually defined somewhere else (see [Chapter 5 \[Record Descriptors\]](#), page 7). The second component is the name of a field. Finally, an optional third component is the role played by the referenced field in the referencing record.

The simplest kind of reference is one without an explicit role. It is used when only one association between records of different types is desired. Consider for example:

```
%rec: Maintainer
```

```
Name: Jose E. Marchesi
Package:Name: recutils
```

In the previous example it is pretty clear which role is played by the referenced package in the `Maintainer` record: it is the package maintained by the maintainer. But sometimes the role is not that identifiable. Consider for example the following record:

```
%rec: Bug
```

```
Id: 203
Title: recsel crashes with files bigger than 2gb
Hacker:Name: Smitha Johnson
```

Seems to be a bug report. But, what is the relationship between Smitha Johnson and the bug? Is she the reporter? Or perhaps she is the hacker that is working to fix it? Is she the hacker that closed the bug? We can clarify the role of Smitha by using the third part of the compound field: her role.

```
Id: 203
Title: recsel crashes with files bigger than 2gb
Hacker:Name:Reporter: Smitha Johnson
```

So now the situation is clear: Smitha is the reporter.

It is possible to make references to different instances of the same record type by using roles. Supposing that we want to record who is the assignee of the bug report, we can introduce the new role `Assignee`:

```
Id: 203
Title: recsel crashes with files bigger than 2gb
Hacker:Name:Reporter: Smitha Johnson
Hacker:Name:Assignee: Juan Valdes
```

Note that, like in the case of regular fields, there can be several compound fields with the same name and the same implicit role in a record, like in:

```
%rec: Maintainer

Name: Jose E. Marchesi
Package:Name: GNU PDF
Package:Name: GNU Ferret
```

```
Package:Name: GNU rec
```

Same applies to several compound fields with explicit roles:

```
%rec: Package
```

```
Name: GNU foo
```

```
Hacker:Name:Maintainer: Elvis 'The King' Presley
```

```
Hacker:Name:Maintainer: Michael 'The other King' Jackson
```

16 Date input formats

First, a quote:

Our units of temporal measurement, from seconds on up to months, are so complicated, asymmetrical and disjunctive so as to make coherent mental reckoning in time all but impossible. Indeed, had some tyrannical god contrived to enslave our minds to time, to make it all but impossible for us to escape subjection to sodden routines and unpleasant surprises, he could hardly have done better than handing down our present system. It is like a set of trapezoidal building blocks, with no vertical or horizontal surfaces, like a language in which the simplest thought demands ornate constructions, useless particles and lengthy circumlocutions. Unlike the more successful patterns of language and science, which enable us to face experience boldly or at least level-headedly, our system of temporal calculation silently and persistently encourages our terror of time.

. . . It is as though architects had to measure length in feet, width in meters and height in ells; as though basic instruction manuals demanded a knowledge of five different languages. It is no wonder then that we often look into our own immediate past or future, last Tuesday or a week from Sunday, with feelings of helpless confusion. . . .

— Robert Grudin, *Time and the Art of Living*.

This section describes the textual date representations that GNU programs accept. These are the strings you, as a user, can supply as arguments to the various programs. The C interface (via the `parse_datetime` function) is not described here.

16.1 General date syntax

A *date* is a string, possibly empty, containing many items separated by whitespace. The whitespace may be omitted when no ambiguity arises. The empty string means the beginning of today (i.e., midnight). Order of the items is immaterial. A date string may contain many flavors of items:

- calendar date items
- time of day items
- time zone items
- combined date and time of day items
- day of the week items
- relative items
- pure numbers.

We describe each of these item types in turn, below.

A few ordinal numbers may be written out in words in some contexts. This is most useful for specifying day of the week items or relative items (see below). Among the most commonly used ordinal numbers, the word ‘**last**’ stands for -1 , ‘**this**’ stands for 0 , and ‘**first**’ and ‘**next**’ both stand for 1 . Because the word ‘**second**’ stands for the unit of time there is no way to write the ordinal number 2 , but for convenience ‘**third**’ stands for 3 , ‘**fourth**’ for 4 , ‘**fifth**’ for 5 , ‘**sixth**’ for 6 , ‘**seventh**’ for 7 , ‘**eighth**’ for 8 , ‘**ninth**’ for 9 , ‘**tenth**’ for 10 , ‘**eleventh**’ for 11 and ‘**twelfth**’ for 12 .

When a month is written this way, it is still considered to be written numerically, instead of being “spelled in full”; this changes the allowed strings.

In the current implementation, only English is supported for words and abbreviations like ‘**AM**’, ‘**DST**’, ‘**EST**’, ‘**first**’, ‘**January**’, ‘**Sunday**’, ‘**tomorrow**’, and ‘**year**’.

The output of the `date` command is not always acceptable as a date string, not only because of the language problem, but also because there is no standard meaning for time zone items like ‘IST’. When using `date` to generate a date string intended to be parsed later, specify a date format that is independent of language and that does not use time zone items other than ‘UTC’ and ‘Z’. Here are some ways to do this:

```
$ LC_ALL=C TZ=UTC0 date
Mon Mar  1 00:21:42 UTC 2004
$ TZ=UTC0 date +%Y-%m-%d %H:%M:%SZ
2004-03-01 00:21:42Z
$ date --rfc-3339=ns # --rfc-3339 is a GNU extension.
2004-02-29 16:21:42.692722128-08:00
$ date --rfc-2822 # a GNU extension
Sun, 29 Feb 2004 16:21:42 -0800
$ date +%Y-%m-%d %H:%M:%S %z # %z is a GNU extension.
2004-02-29 16:21:42 -0800
$ date +%@%s.%N # %s and %N are GNU extensions.
@1078100502.692722128
```

Alphabetic case is completely ignored in dates. Comments may be introduced between round parentheses, as long as included parentheses are properly nested. Hyphens not followed by a digit are currently ignored. Leading zeros on numbers are ignored.

Invalid dates like ‘2005-02-29’ or times like ‘24:00’ are rejected. In the typical case of a host that does not support leap seconds, a time like ‘23:59:60’ is rejected even if it corresponds to a valid leap second.

16.2 Calendar date items

A *calendar date item* specifies a day of the year. It is specified differently, depending on whether the month is specified numerically or literally. All these strings specify the same calendar date:

```
1972-09-24 # ISO 8601.
72-9-24 # Assume 19xx for 69 through 99,
# 20xx for 00 through 68.
72-09-24 # Leading zeros are ignored.
9/24/72 # Common U.S. writing.
24 September 1972
24 Sept 72 # September has a special abbreviation.
24 Sep 72 # Three-letter abbreviations always allowed.
Sep 24, 1972
24-sep-72
24sep72
```

The year can also be omitted. In this case, the last specified year is used, or the current year if none. For example:

```
9/24
sep 24
```

Here are the rules.

For numeric months, the ISO 8601 format ‘*year-month-day*’ is allowed, where *year* is any positive number, *month* is a number between 01 and 12, and *day* is a number between 01 and 31. A leading zero must be present if a number is less than ten. If *year* is 68 or smaller, then 2000 is added to it; otherwise, if *year* is less than 100, then 1900 is added to it. The construct ‘*month/day/year*’, popular in the United States, is accepted. Also ‘*month/day*’, omitting the year.

Literal months may be spelled out in full: ‘January’, ‘February’, ‘March’, ‘April’, ‘May’, ‘June’, ‘July’, ‘August’, ‘September’, ‘October’, ‘November’ or ‘December’. Literal months may be abbreviated to their first three letters, possibly followed by an abbreviating dot. It is also permitted to write ‘Sept’ instead of ‘September’.

When months are written literally, the calendar date may be given as any of the following:

```
day month year
day month
month day year
day-month-year
```

Or, omitting the year:

```
month day
```

16.3 Time of day items

A *time of day item* in date strings specifies the time on a given day. Here are some examples, all of which represent the same time:

```
20:02:00.000000
20:02
8:02pm
20:02-0500      # In EST (U.S. Eastern Standard Time).
```

More generally, the time of day may be given as ‘*hour:minute:second*’, where *hour* is a number between 0 and 23, *minute* is a number between 0 and 59, and *second* is a number between 0 and 59 possibly followed by ‘.’ or ‘,’ and a fraction containing one or more digits. Alternatively, ‘:*second*’ can be omitted, in which case it is taken to be zero. On the rare hosts that support leap seconds, *second* may be 60.

If the time is followed by ‘am’ or ‘pm’ (or ‘a.m.’ or ‘p.m.’), *hour* is restricted to run from 1 to 12, and ‘:*minute*’ may be omitted (taken to be zero). ‘am’ indicates the first half of the day, ‘pm’ indicates the second half of the day. In this notation, 12 is the predecessor of 1: midnight is ‘12am’ while noon is ‘12pm’. (This is the zero-oriented interpretation of ‘12am’ and ‘12pm’, as opposed to the old tradition derived from Latin which uses ‘12m’ for noon and ‘12pm’ for midnight.)

The time may alternatively be followed by a time zone correction, expressed as ‘*s hhmm*’, where *s* is ‘+’ or ‘-’, *hh* is a number of zone hours and *mm* is a number of zone minutes. The zone minutes term, *mm*, may be omitted, in which case the one- or two-digit correction is interpreted as a number of hours. You can also separate *hh* from *mm* with a colon. When a time zone correction is given this way, it forces interpretation of the time relative to Coordinated Universal Time (UTC), overriding any previous specification for the time zone or the local time zone. For example, ‘+0530’ and ‘+05:30’ both stand for the time zone 5.5 hours ahead of UTC (e.g., India). This is the best way to specify a time zone correction by fractional parts of an hour. The maximum zone correction is 24 hours.

Either ‘am’/‘pm’ or a time zone correction may be specified, but not both.

16.4 Time zone items

A *time zone item* specifies an international time zone, indicated by a small set of letters, e.g., ‘UTC’ or ‘Z’ for Coordinated Universal Time. Any included periods are ignored. By following a non-daylight-saving time zone by the string ‘DST’ in a separate word (that is, separated by some white space), the corresponding daylight saving time zone may be specified. Alternatively, a non-daylight-saving time zone can be followed by a time zone correction, to add the two values. This is normally done only for ‘UTC’; for example, ‘UTC+05:30’ is equivalent to ‘+05:30’.

Time zone items other than ‘UTC’ and ‘Z’ are obsolescent and are not recommended, because they are ambiguous; for example, ‘EST’ has a different meaning in Australia than in the United States. Instead, it’s better to use unambiguous numeric time zone corrections like ‘-0500’, as described in the previous section.

If neither a time zone item nor a time zone correction is supplied, time stamps are interpreted using the rules of the default time zone (see [Section 16.10 \[Specifying time zone rules\]](#), page 30).

16.5 Combined date and time of day items

A *combined date and time of day item* specifies the time on a specific day of the year. This type is needed for formats that cannot be represented by individual calendar date (see [Section 16.2 \[Calendar date items\]](#), page 26) and time of day (see [Section 16.3 \[Time of day items\]](#), page 27) items due to ambiguity.

```
# ISO 8601 extended date and time of day format
1972-09-24T20:02:00,000000-0500
```

The ISO 8601 extended date and time of day format is an ISO 8601 date, a ‘T’ character separator, followed by an ISO 8601 time of day.

16.6 Day of week items

The explicit mention of a day of the week will forward the date (only if necessary) to reach that day of the week in the future.

Days of the week may be spelled out in full: ‘Sunday’, ‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’, ‘Friday’ or ‘Saturday’. Days may be abbreviated to their first three letters, optionally followed by a period. The special abbreviations ‘Tues’ for ‘Tuesday’, ‘Wednes’ for ‘Wednesday’ and ‘Thur’ or ‘Thurs’ for ‘Thursday’ are also allowed.

A number may precede a day of the week item to move forward supplementary weeks. It is best used in expression like ‘third monday’. In this context, ‘last day’ or ‘next day’ is also acceptable; they move one week before or after the day that *day* by itself would represent.

A comma following a day of the week item is ignored.

16.7 Relative items in date strings

Relative items adjust a date (or the current date if none) forward or backward. The effects of relative items accumulate. Here are some examples:

```
1 year
1 year ago
3 years
2 days
```

The unit of time displacement may be selected by the string ‘year’ or ‘month’ for moving by whole years or months. These are fuzzy units, as years and months are not all of equal duration. More precise units are ‘fortnight’ which is worth 14 days, ‘week’ worth 7 days, ‘day’ worth 24 hours, ‘hour’ worth 60 minutes, ‘minute’ or ‘min’ worth 60 seconds, and ‘second’ or ‘sec’ worth one second. An ‘s’ suffix on these units is accepted and ignored.

The unit of time may be preceded by a multiplier, given as an optionally signed number. Unsigned numbers are taken as positively signed. No number at all implies 1 for a multiplier. Following a relative item by the string ‘ago’ is equivalent to preceding the unit by a multiplier with value -1 .

The string ‘tomorrow’ is worth one day in the future (equivalent to ‘day’), the string ‘yesterday’ is worth one day in the past (equivalent to ‘day ago’).

The strings ‘now’ or ‘today’ are relative items corresponding to zero-valued time displacement, these strings come from the fact a zero-valued time displacement represents the current time when not otherwise changed by previous items. They may be used to stress other items, like in ‘12:00 today’. The string ‘this’ also has the meaning of a zero-valued time displacement, but is preferred in date strings like ‘this thursday’.

When a relative item causes the resulting date to cross a boundary where the clocks were adjusted, typically for daylight saving time, the resulting date and time are adjusted accordingly.

The fuzz in units can cause problems with relative items. For example, ‘2003-07-31 -1 month’ might evaluate to 2003-07-01, because 2003-06-31 is an invalid date. To determine the previous month more reliably, you can ask for the month before the 15th of the current month. For example:

```
$ date -R
Thu, 31 Jul 2003 13:02:39 -0700
$ date --date='-1 month' +'Last month was %B?'
Last month was July?
$ date --date="$(date +%Y-%m-15) -1 month" +'Last month was %B!'
Last month was June!
```

Also, take care when manipulating dates around clock changes such as daylight saving leaps. In a few cases these have added or subtracted as much as 24 hours from the clock, so it is often wise to adopt universal time by setting the TZ environment variable to ‘UTC0’ before embarking on calendrical calculations.

16.8 Pure numbers in date strings

The precise interpretation of a pure decimal number depends on the context in the date string.

If the decimal number is of the form *yyyymmdd* and no other calendar date item (see [Section 16.2 \[Calendar date items\], page 26](#)) appears before it in the date string, then *yyyy* is read as the year, *mm* as the month number and *dd* as the day of the month, for the specified calendar date.

If the decimal number is of the form *hhmm* and no other time of day item appears before it in the date string, then *hh* is read as the hour of the day and *mm* as the minute of the hour, for the specified time of day. *mm* can also be omitted.

If both a calendar date and a time of day appear to the left of a number in the date string, but no relative item, then the number overrides the year.

16.9 Seconds since the Epoch

If you precede a number with ‘@’, it represents an internal time stamp as a count of seconds. The number can contain an internal decimal point (either ‘.’ or ‘,’); any excess precision not supported by the internal representation is truncated toward minus infinity. Such a number cannot be combined with any other date item, as it specifies a complete time stamp.

Internally, computer times are represented as a count of seconds since an epoch—a well-defined point of time. On GNU and POSIX systems, the epoch is 1970-01-01 00:00:00 UTC, so ‘@0’ represents this time, ‘@1’ represents 1970-01-01 00:00:01 UTC, and so forth. GNU and most other POSIX-compliant systems support such times as an extension to POSIX, using negative counts, so that ‘@-1’ represents 1969-12-31 23:59:59 UTC.

Traditional Unix systems count seconds with 32-bit two’s-complement integers and can represent times from 1901-12-13 20:45:52 through 2038-01-19 03:14:07 UTC. More modern systems use 64-bit counts of seconds with nanosecond subcounts, and can represent all the times in the known lifetime of the universe to a resolution of 1 nanosecond.

On most hosts, these counts ignore the presence of leap seconds. For example, on most hosts ‘@915148799’ represents 1998-12-31 23:59:59 UTC, ‘@915148800’ represents 1999-01-01 00:00:00 UTC, and there is no way to represent the intervening leap second 1998-12-31 23:59:60 UTC.

16.10 Specifying time zone rules

Normally, dates are interpreted using the rules of the current time zone, which in turn are specified by the TZ environment variable, or by a system default if TZ is not set. To specify a different set of default time zone rules that apply just to one date, start the date with a string of the form ‘TZ=*rule*’. The two quote characters (“”) must be present in the date, and any quotes or backslashes within *rule* must be escaped by a backslash.

For example, with the GNU `date` command you can answer the question “What time is it in New York when a Paris clock shows 6:30am on October 31, 2004?” by using a date beginning with ‘TZ="Europe/Paris"' as shown in the following shell transcript:

```
$ export TZ="America/New_York"
$ date --date='TZ="Europe/Paris" 2004-10-31 06:30'
Sun Oct 31 01:30:00 EDT 2004
```

In this example, the ‘--date’ operand begins with its own TZ setting, so the rest of that operand is processed according to ‘Europe/Paris’ rules, treating the string ‘2004-10-31 06:30’ as if it were in Paris. However, since the output of the `date` command is processed according to the overall time zone rules, it uses New York time. (Paris was normally six hours ahead of New York in 2004, but this example refers to a brief Halloween period when the gap was five hours.)

A TZ value is a rule that typically names a location in the ‘tz’ database. A recent catalog of location names appears in the [TWiki Date and Time Gateway](#). A few non-GNU hosts require a colon before a location name in a TZ setting, e.g., ‘TZ=":America/New_York"’.

The ‘tz’ database includes a wide variety of locations ranging from ‘Arctic/Longyearbyen’ to ‘Antarctica/South_Pole’, but if you are at sea and have your own private time zone, or if you are using a non-GNU host that does not support the ‘tz’ database, you may need to use a POSIX rule instead. Simple POSIX rules like ‘UTC0’ specify a time zone without daylight saving time; other rules can specify simple daylight saving regimes. See [Section “Specifying the Time Zone with TZ”](#) in *The GNU C Library*.

16.11 Authors of parse_datetime

`parse_datetime` started life as `getdate`, as originally implemented by Steven M. Bellovin (smb@research.att.com) while at the University of North Carolina at Chapel Hill. The code was later tweaked by a couple of people on Usenet, then completely overhauled by Rich Salz (rsalz@bbn.com) and Jim Berets (jberets@bbn.com) in August, 1990. Various revisions for the GNU system were made by David MacKenzie, Jim Meyering, Paul Eggert and others, including renaming it to `get_date` to avoid a conflict with the alternative Posix function `getdate`, and a later rename to `parse_datetime`. The Posix function `getdate` can parse more locale-specific dates using `strptime`, but relies on an environment variable and external file, and lacks the thread-safety of `parse_datetime`.

This chapter was originally produced by François Pinard (pinard@iro.umontreal.ca) from the ‘`parse_datetime.y`’ source code, and then edited by K. Berry (kb@cs.umb.edu).

17 Regular Expressions

The character `'.'` matches any single character except the null character.

- `'+'` indicates that the regular expression should match one or more occurrences of the previous atom or regexp.
- `'?'` indicates that the regular expression should match zero or one occurrence of the previous atom or regexp.
- `'\+'` matches a `'+'`
- `'\?'` matches a `'?'`.

Bracket expressions are used to match ranges of characters. Bracket expressions where the range is backward, for example `'[z-a]'`, are invalid. Within square brackets, `'\'` is taken literally. Character classes are supported; for example `'[[:digit:]]'` will match a single decimal digit.

GNU extensions are supported:

1. `'\w'` matches a character within a word
2. `'\W'` matches a character which is not within a word
3. `'\<'` matches the beginning of a word
4. `'\>'` matches the end of a word
5. `'\b'` matches a word boundary
6. `'\B'` matches characters which are not a word boundary
7. `'\''` matches the beginning of the whole input
8. `'\''` matches the end of the whole input

Grouping is performed with parentheses `'()'`. An unmatched `')'` matches just itself. A backslash followed by a digit acts as a back-reference and matches the same thing as the previous grouped expression indicated by that number. For example `'\2'` matches the second group expression. The order of group expressions is determined by the position of their opening parenthesis `'('`.

The alternation operator is `'|'`.

The characters `'^'` and `'$'` always represent the beginning and end of a string respectively, except within square brackets. Within brackets, `'^'` can be used to invert the membership of the character class being specified.

`'*'`, `'+'` and `'?'` are special at any point in a regular expression except the following places, where they are not allowed:

1. At the beginning of a regular expression
2. After an open-group, signified by `'('`
3. After the alternation operator `'|'`

Intervals are specified by `'{'` and `'}'`. Invalid intervals such as `'a{1z}'` are not accepted.

The longest possible match is returned; this applies to the regular expression as a whole and (subject to this constraint) to subexpressions within groups.

18 Common Options

Certain options are available in all of these programs. Rather than writing identical descriptions for each of the programs, they are described here.

- '`--version`' Print the version number, then exit successfully.
- '`--help`' Print a help message, then exit successfully.
- '`--`' Delimit the option list. Later arguments, if any, are treated as operands even if they begin with '-'. For example, `recsel -- -p` reads from the file named '`r`'.

18.1 Selection Expressions

Selection expressions (also known as a SEXs) are simple infix expressions that can be applied to a record. The result of the SEX is typically interpreted as a Boolean value.

18.1.1 Operands

The supported operands are: numbers, strings, field names and parenthesized expressions.

18.1.1.1 Numeric Literals

The supported numeric literals are integer numbers and real numbers. The usual sign character '-' is used to denote negative values. Integer values can be denoted in base 10, base 16 using the 0x prefix, and base 8 using the 0 prefix. Examples are:

```
10000
0
0xFF
-0xa
012
-07
-1342
.12
-3.14
```

18.1.1.2 String Literals

String values are delimited by either the ' character or the " character. Whatever delimiter is used, the delimiter closing the literal shall equal to the delimiter used to open it.

Note that newlines and tabs can be part of a string literal.

Examples are:

```
'Hello.'
```

```
'The following example is the empty string.'
```

```
''
```

The ' and " characters can be part of a string if they are escaped with a backslash, like in:

```
'This string contains an apostrophe: \'.'
```

```
"This one a double quote: \"."
```

18.1.1.3 Field Values

The value of a field value can be included in a selection expression by writing its name. The field name is replaced by a string containing the field value, covering any possibility with records with more than one field featuring that name. The last colon character (:) of the field name is optional. Examples:

```
Name
Email:
Hacker:Name:OpenedBy
```

It is possible to use the role part of a field if it is not empty. So, for example, if we are searching for the issues opened by 'John Smith' in a database of issues we could write:

```
$ recsel -e "OpenedBy = 'John Smith'"
```

instead of using the full field name:

```
$ recsel -e "Hacker:Name:OpenedBy = 'John Smith'"
```

When the name of a field appears in an expression, the expression is applied to all the fields in the record featuring that name. So, for example, the expression:

```
Email ~ "\\\.org"
```

Will match any record in which there is a field named 'Email' whose value terminates in '.org'. If we are interested in the value of some specific email, we can specify its relative position into the containing record by using *subscripts*. Consider, for example:

```
Email[0] ~ "\\\.org"
```

Will match for:

```
Name: Mr. Foo
Email: foo@foo.org
Email: mr.foo@foo.com
```

But not for:

```
Name: Mr. Foo
Email: mr.foo@foo.com
Email: foo@foo.org
```

The regexp flavor supported in selection expressions are the POSIX EREs plus several GNU extensions. See [Chapter 17 \[Regular Expressions\], page 31](#).

18.1.1.4 Parenthesized Expressions

Parenthesis characters ((and)) can be used to group sub expressions in the usual way.

18.1.2 Operators

The supported operators are arithmetic operators (addition, subtraction, multiplication, division and modulus), logical operators, string operators and field operators.

18.1.2.1 Arithmetic Operators

Arithmetic operators for addition (+), subtraction (-), multiplication (*), integer division (/) and modulus (%) are supported with their usual meanings.

These operators require either numeric operands or string operands whose value can be interpreted as numbers (integer or real).

18.1.2.2 Boolean Operators

The boolean operators **and** (&&), **or** (||) and **not** (!) are supported with the same semantics as their C counterparts.

The boolean operators expect integer operands, and will try to convert any string operand to an integer value.

18.1.2.3 Comparison Operators

The compare operators **less than** (<), **greater than** (>), **less than or equal** (<=), **greater than or equal** (>=), **equal** (=) and **unequal** (!=) are supported with their usual meaning.

Strings can be compared with the equality operator (=).

The match operator (~) can be used to match a string with a given regular expression. The supported regexp syntax is described in the GNU C library manual.

18.1.2.4 Date Comparison Operators

The compare operators **before** (<<), **after** (>>) and **same time** (==) can be used with fields and strings containing parseable dates.

See [Chapter 16 \[Date input formats\]](#), page 25.

18.1.2.5 Field Operators

Field counters are replaced by the number of occurrences of a field with the given name in the record. For example:

```
#Email
```

The previous expression is replaced with the number of fields named **Email** in the record. It can be zero if the record does not have a field with that name.

18.1.2.6 String Operators

The string concatenation operator (&) can be used to concatenate any number of strings and field values.

```
'foo' & Name & 'bar'
```

18.1.2.7 Conditional Operator

The ternary conditional operator can be used to select alternatives based on the value of some expression:

```
expr1 ? expr2 : expr3
```

If **expr1** evaluates to true (i.e. it is an integer or the string representation of an integer and its value is not zero) then the operator yields **expr2**. Otherwise it yields **expr3**.

Note that there should be always at least one blank character between **expr2** and the colon (:). If **expr2** is a field name. This is because field names can optionally end with a colon.

18.1.3 Evaluation of Selection Expressions

Given that:

- It is possible to refer to fields by name in selection expressions.
- Records can have several fields featuring the same name.

It is clear that some backtracking mechanism is needed in the evaluation of the selection expressions. For example, consider the following expression that is deciding whether a “registration” in a webpage shall be rejected:

```
((Email ~ "foomail\.com") || (Age <= 18)) && !#Fixed
```

The previous expression will be evaluated for every possible permutation of the fields “Email”, “Age” and “Fixed” present in the record, until one of the combinations succeeds. At that point the computation is interrupted.

When used to decide whether a record matches some criteria, the goal of a selection expression is to act as a boolean expression. In that case the final value of the expression depends on both the type and the value of the result launched by the top-most subexpression:

- If the result is an **integer**, the expression is true if its value is not zero.
- If the result is a **real**, or a **string**, the expression evaluates to false.

Sometimes a selection expression is used to compute a result instead of a boolean. In that case the returned value is converted to a string. This is used when replacing the slots in templates (see [Section 25.2 \[recfmt Templates\]](#), page 48).

18.2 Field Expressions

Field expressions (also known as FEXs) are a way to select fields of a record.

A FEX is composed by a sequence of *elements* separated by commas:

`ELEM_1,ELEM_2,...,ELEM_N`

Each element makes a reference to one or more fields in a record identified by a given name and an optional subscript:

`Field_Name [min-max]`

min and *max* are zero-based indexes. It is possible to refer to a field occupying a given position. For example, consider the following record:

```
Name: Mr. Foo
Email: foo@foo.com
Email: foo@foo.org
Email: mr.foo@foo.org
```

We would select all the emails of the record with:

`Email`

The first email with:

`Email[0]`

The third email with:

`Email[2]`

The second and the third email with:

`Email[1-2]`

And so on. Note that a selection is an ordered set not allowing duplicated values. Thus, the field expression:

`Email[0],Name,Email`

is equivalent to

`Email[0],Name,Email[1-2]`

19 recinf

`recinf` reads the given rec files (or the data in the standard input if no file is specified) and prints information about it.

19.1 recinf Invocation

`recinf` reads the given rec files (or the data in the standard input if no file is specified) and prints a resume of the record types contained in the input.

Synopsis:

```
recinf [option]... [file]...
```

The default behavior of the tool is to emit a line per record type in the input containing its name and the number of records of that type:

```
$ recinf hackers.rec tasks.rec
25 Hacker
102 Task
```

If the input contains anonymous records, i.e. records that are before the first record descriptor, the corresponding line resume won't have a type name:

```
$ recinf data.rec
10
```

In addition of the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

'-t TYPE'

'--type=type'

Select records of a given type only.

'-d'

'--descriptor'

Print all the record descriptors present in the file.

'-n'

'--names-only'

Output just the names of the record types found in the input. If the input is composed by anonymous records only then don't emit any output.

'--print-sexps'

Print the data in the form of sexps (lisp expressions) instead of rec format. This option is intended to be used by lisp programs.

20 recsel

`recsel` reads the given rec files (or the data in the standard input if no file is specified) and prints out records (or part of records) based in some criteria specified by the user.

20.1 recsel Invocation

`recsel` searches rec files for records satisfying a certain criteria. Synopsis:

```
recsel [option]... [-n indexes | -e record_expr | -q str | -m num] [-c | (-p|-P|-R) f
      [file]...
```

In addition of the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

If no FILE is specified then the command acts like a filter, getting the data from the standard input and writing the result in the standard output.

The following *global options* are available.

‘-C’

‘--collapse’

Do not section the result in records with newlines.

‘-d’

‘--include-descriptors’

Print record descriptors along with the matched records

‘-s secret’

‘--password=secret’

Try to decrypt confidential fields with the given password.

‘-S’

‘--sort=field’

Sort the output by *field*. This option has precedence to whatever sorting criteria is specified in the corresponding record descriptor with `%sort`.

‘-U’

‘--uniq’ Remove duplicated fields in the output records. Fields are duplicated if there are more than one featuring the same field name and the same value.

The *selection options* are used to select a subset of the records in the input.

‘-n indexes’

Match the records occupying the given positions in its record set. *indexes* must be a comma-separated list of numbers or ranges, the ranges being two numbers separated with dashes. For example, the following list denotes the first, the third, the fourth and all records up to the tenth: `-n 0,2,4-9`.

‘-e expr’

‘--expression=expr’

A record selection expression (see [Section 18.1 \[Selection Expressions\]](#), [page 32](#)). Only the records matched by the expression will be taken into account to compute the output.

‘-q str’

‘--quick=str’

Select records having a field whose value contains the substring *str*.

'-m *num*'
 '--random=*num*'
 Select *num* random records. If *num* is zero then select all the records.

'-t TYPE'
 '--type=*type*'
 Select records of a given type only.

The *output options* are used to determine what information about the selected records to display to the user, and how to display it.

'-p *name_list*'
 '--print=*name_list*'
 List of fields to print for each record. *name_list* is a list of field names separated by commas. For example:

```
-p Name,Email
```

means to print the Name and the Email of every matching record.

If this option is not specified then all the fields of the matching records are printed in the standard output.

'-P *name_list*'
 '--print-values=*name_list*'
 Same than '-p', but print the values of the selected fields.

'-R *name_list*'
 '--print-row=*name_list*'
 Same than '-P', but print the values separated by single spaces instead of newlines.

'-c'
 '--count' If this option is specified then recsel will print the number of matching records instead of the records themselves. This option is incompatible with '-p', '-P' and '-R'.

Some *special options* are available to ease the communication between the recutils and other programs, such as lisp interpreters. Those options are not intended to be used by human operators.

'--print-sexps'
 Print the data using sexps instead of rec format.

20.2 recsel Examples

Print the closed bugs:

```
$ recsel -t Task -e "Status = 'CLOSED'" TODO.rec
```

Print the name of all the registrants less than twenty:

```
$ recsel -e 'Age < 20' -P Partner registrants.rec
```

20.3 recsel Encryption

The contents of confidential fields can be read using the '-s|--password' command line option to `recsel`. When used, any selected record containing encrypted fields will try to decrypt them with the given password. If the operation succeeds then the output will include the unencrypted data. Otherwise the ASCII-encoded encrypted data will be emitted.

If `recsel` is invoked interactively and no password is specified with '-s', the user will be asked for a password in case one is needed. No echo of the password will appear in the screen.

The provided password will be used to decrypt all confidential fields as if it was specified with ‘-s’.

For example, consider the following database storing information about the user accounts of some online service. Each entry stores a login, a full name, email and a password. The password is declared as confidential:

```
%rec: Account
%key: Login
%confidential: Password

Login: foo
Name: Mr. Foo
Email: foo@foo.com
Password: encrypted-AAABBBCCCDDD

Login: bar
Name: Ms. Bar
Email: bar@bar.org
Password: encrypted-XXXYYYYZZZUUU
```

If we use `recsel` to get a list of records of type `Account` without specifying a password, or if the wrong password was specified in interactive mode, then we would get the following output with the encrypted values:

```
$ cat accounts.rec | recsel -t Account -p Login,Password
Login: foo
Password: encrypted-AAABBBCCCDDD

Login: bar
Password: encrypted-XXXYYYYZZZUUU
```

If we specify a password and both entries were encrypted using that password, we would get the unencrypted values:

```
$ recsel -t Account -s secret -p Login,Password accounts.rec
Login: foo
Password: foosecret

Login: bar
Password: barsecret
```

Note that nothing prevents to have confidential fields encrypted with different passwords. As discussed in see [Chapter 14 \[%confidential\], page 21](#) this can be useful to implement several “levels” of security. For example, we may have an entry in our database with data about the account of the administrator of the online service. In that case we could want to store the password associated with that account using a differentiated password. In that case the output of the last command would have been:

```
$ recsel -t Account -s secret -p Login,Password accounts.rec
Login: foo
Password: foosecret

Login: bar
Password: barsecret

Login: admin
Password: encrypted-TTTVVVBBBNNN
```

We would need to invoke `recsel` with the password used to encrypt the admin entry in order to read it back unencrypted.

21 recins

`recins` adds new records to a rec file or to rec data read from the standard input.

21.1 recins Invocation

`recins` adds new records to a rec file or to rec data read from the standard input. Synopsis:

```
recins [option]... [t type] [-n indexes | -e expr | -q str | -m num] \
      [(-f str -v str)|[-r recdata)]... [file]
```

The new record that will be inserted by the command is constructed by using pairs of ‘`-f`’ and ‘`-v`’ options. Each pair defines a field. The order of the parameters is significant.

If no FILE is specified then the command acts like a filter, getting the data from the standard input and writing the result in the standard output.

If the specified FILE does not exist, it is created.

In addition to the common options described earlier (see [Chapter 18 \[Common Options\], page 32](#)) the program accepts the following options.

‘`-t`’

‘`--type=expr`’

The type of the new record. If there is a record set in the input data matching this type then the new record is added there. Otherwise a new record set is created. If this parameter is not specified then the new record is anonymous.

‘`-f`’

‘`--field=name`’

Declares the name of a field. This option shall be followed by a ‘`-v`’.

‘`-v`’

‘`--value=value`’

The value of the field being defined.

‘`-r`’

‘`--record=value`’

Add the fields of the record in *value*. This option can be intermixed with `-f ... -v` pairs.

‘`-s`’

‘`--password`’

Encrypt confidential fields with the given password.

‘`--no-external`’

Don’t use external record descriptors.

‘`--verbose`’

Be verbose when reporting integrity problems.

‘`--no-auto`’

Don’t generate *auto* fields. See [Chapter 11 \[%auto\], page 17](#).

Record selection arguments are supported as well. If they are used then `recins` enters in “replacement mode”. Instead of appending the new record, matched records are replaced by copies of the provided record. The selection arguments are summarized in the next table.

‘`-n indexes`’

Match the records occupying the given positions in its record set. *indexes* must be a comma-separated list of numbers or ranges, the ranges being two numbers separated with dashes. For example, the following list denotes the first, the third, the fourth and all records up to the tenth: `-n 0,2,4-9`.

```

'-e expr'
'--expression=expr'
    A record selection expression (see Section 18.1 \[Selection Expressions\], page 32).
    Matching records will get replaced.

'-q str'
'--quick=str'
    Remove records having a field whose value contains the substring str.

'-m num'
'--random=num'
    Select num random records. If num is zero then all records are selected, i.e. no
    replace mode is activated.

'-i'
'--case-insensitive'
    Make strings case-insensitive in selection expressions.

```

21.2 recins Examples

Create a new issue in the bugs database:

```

$ recins -t Task -f Id -v 10 \
          -f Title -v "New issue." \
          -f Status -v NEW \
          TODO.rec

```

Register a new event in a log file, using recins as a filter:

```

recins -f Date -v 'date' -f Entry -v "$HW_ADDR device connected" \
      < $LOGFILE > $LOGFILE.t \
      && cat $LOGFILE.t >> $LOGFILE

```

21.3 recins Encryption

`recins` allows the insertion of encrypted fields in a database. When the `'-s|--password'` command line option is specified in the command line any field declared as confidential in the record descriptor will get encrypted using the given passphrase. If the command is executed interactively and `'-s'` is not used then the user is asked to provide a password using the terminal. For example, the invocation:

```

$ recins -t Account -s mypassword -f Login -v foo -f Password -v secret accounts.rec

```

Will encrypt the value of the `Password` field with `mypassword` as long as the field is declared as confidential. See [Chapter 14 \[%confidential\]](#), page 21 for details on confidential fields.

`recins` will issue a warning if a confidential field is inserted in the database but no password was provided to encrypt it. This is to avoid having unencrypted sensible data in the recfiles.

22 recdel

`recdel` removes records from a rec file, or from rec data read from the standard input.

22.1 recdel Invocation

`recdel` removes records from a rec file, or from rec data read from the standard input. Synopsis:

```
recdel [OPTIONS]... [-t type] [-n indexes | -e expr | -q str | -m num] [file]
```

If no *file* is specified then the command acts like a filter, getting the data from the standard input and writing the result in the standard output.

In addition to the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

'-t'

'--type=*expr*'

Remove records of the given type. If this parameter is not specified then records of any type will be removed.

'-n *indexes*'

Match the records occupying the given positions in its record set. *indexes* must be a comma-separated list of numbers or ranges, the ranges being two numbers separated with dashes. For example, the following list denotes the first, the third, the fourth and all records up to the tenth: `-n 0,2,4-9`.

'-e *expr*'

'--expression=*expr*'

A record selection expression (see [Section 18.1 \[Selection Expressions\]](#), [page 32](#)). Only the records matched by the expression will be removed from the file.

'-q *str*'

'--quick=*str*'

Remove records having a field whose value contains the substring *str*.

'-m *num*'

'--random=*num*'

Remove *num* random records. If *num* is zero then remove all the records.

'-c'

'--comment'

Comment the matching records out instead of removing them.

'-f'

'--force' Delete even in potentially dangerous situations, such as the request to delete all the records of some type, for example.

'--no-external'

Don't use external record descriptors.

'--verbose'

Be verbose when reporting integrity problems.

22.2 recdel Examples

Comment out closed issues in the bugs database:

```
$ recdel -c -t Task -e "Status = 'CLOSED'" TODO.rec
```

23 `recset`

`recset` manipulates the fields of records in a `rec` file.

23.1 `recset` Invocation

`recset` manipulates the fields of records in a `rec` file, or `rec` data read from the standard input. Synopsis:

```
recset [option]... [file]...
```

If no *file* is specified then the command acts like a filter, getting the data from the standard input and writing the result in the standard output.

In addition to the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

Record selection options:

```
'-i'
```

```
'--case-insensitive'
```

Make strings case-insensitive in selection expressions.

```
'-t'
```

```
'--type=expr'
```

Operate on the records of the given type. If this parameter is not specified then records of any type will be removed.

```
'-n indexes'
```

Operate on the records occupying the given positions in its record set. *indexes* must be a comma-separated list of numbers or ranges, the ranges being two numbers separated with dashes. For example, the following list denotes the first, the third, the fourth and all records up to the tenth: `-n 0,2,4-9`.

```
'-e expr'
```

```
'--expression=expr'
```

A record selection expression (see [Section 18.1 \[Selection Expressions\]](#), [page 32](#)). Only the records matched by the expression will be processed.

```
'-q str'
```

```
'--quick=str'
```

Operate on records having a field whose value contains the substring *str*.

```
'-m num'
```

```
'--random=num'
```

Operate on *num* random records. If *num* is zero then operate on all the records.

Fields selection options:

```
'-f'
```

```
'--fields=FEX'
```

Field selection expression (see [Section 18.2 \[Field Expressions\]](#), [page 35](#)) to select the fields to operate.

Actions:

```
'-s'
```

```
'--set=VALUE'
```

Set the value of the selected fields to *VALUE*.

`'-a'`
`'--add=VALUE'`
 Add a new field to the selected record with value *VALUE*.

`'-S'`
`'--set-add=VALUE'`
 Set the value of the selected fields to *VALUE*. If some of the fields don't exist in a record, append it with the specified value.

`'-r'`
`'--rename=VALUE'`
 Rename a field. *VALUE* shall be a valid field name. The field expression associated with this action shall contain a single field name and an optional subscript. If an entire record set is selected then the field is renamed in the record descriptor as well.

`'-d'`
`'--delete'`
 Delete the selected fields in the selected records.

`'-c'`
`'--comment'`
 Comment out the selected fields in the selected records.

`'--no-external'`
 Don't use external record descriptors.

`'--verbose'`
 Be verbose when reporting integrity problems.

`'--force'` Perform the requested operation even in potentially dangerous situations, or when the integrity of the data stored in the fail is affected.

23.2 recset Examples

Remove "TmpName" fields from any record in "data.rec":

```
$ recset -f TmpName -d data.rec
```

Set the secondary email of all friends to invalid@email.com:

```
$ recset -f Email[1] -s invalid@email.com friends.rec
```

Add the email new@email.com to John Smith:

```
$ recset -e "Name = 'John Smith'" -f Email -a new@email.com friends.rec
```

Rename the secondary email fields from Email to AltEmail. Modify the record descriptor as well:

```
$ recset -f Email[1] -r AltEmail friends.rec
```

Add a ClosedAt field to the selected record with the current date. In case it already exists, set its value instead:

```
$ recset -n 102 -f ClosedAt -S 'date' tasks.rec
```

24 recfix

recfix checks and fixes rec files.

24.1 recfix Invocation

recfix checks and fixes rec files. Synopsis:

```
recfix [option]... [operation] [op_option]... [file]
```

If no *file* is specified then the command acts like a filter, getting the data from the standard input and writing the result in the standard output.

In addition to the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following global options.

`--no-external`

Don't use external record descriptors.

The effect of running `recfix` depends on the operation it performs. The operation mode is selected by using one of the following options.

`--check` Check the integrity of the database contained in the file, printing diagnostics messages in case something is not right. This is the default operation.

`--sort` Perform a physical sort of all the records contained in the file (or the standard input) after checking for its integrity. The sorting criteria is provided by the `%sort` special field, if any. If there is an integrity failure the sorting is not performed.

Note that this is a destructive operation.

`--encrypt`

Encrypt all the non encrypted fields in the database which are marked as confidential. This operation requires a password. If no password is specified with `-s` and the program is run in a terminal, a prompt is used to get the password from the user.

Note that this is a destructive operation.

`--decrypt`

Decrypt all the encrypted fields in the database which are marked as confidential. This operation requires a password. If no password is specified with `-s` and the program is run in a terminal, a prompt is used to get the password from the user.

Note that this is a destructive operation.

`--auto` Insert auto-generated fields as appropriate in the records which are missing them.

Note that this is a destructive operation.

Some operations make use of certain specific options, which are described in the table below.

`-s secret`

`--password=secret`

Password used to encrypt or decrypt fields.

24.2 recfix Examples

Check and fix a rec database, invoked both destructively and as a filter:

```
$ recfix --check data.rec
$ recfix data.rec
$ cat data.rec | recfix --check data.rec
```

Physically sort the record sets contained in a file:

```
$ recfix --sort data.rec
```

Encrypt any non encrypted confidential fields in a file using a given password:

```
$ recfix --encrypt -s "passphrase" secrets.rec
```

Generate Id entries in a stockage database in records missing them. The record descriptor of the `Item` record set must be defining `Id` as an auto-generated field:

```
$ recfix --auto stockage.rec
```

25 recfmt

recfmt formats records using templates.

25.1 recfmt Invocation

recfmt formats records using templates. Synopsis:

```
recfmt [option]... [template]
```

This program always works as a filter, getting the data from the standard input and writing the result in the standard output.

In addition to the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

```
'-f'
```

```
'--filename=PATH'
```

Read the template from the file in *PATH* instead of the command line.

25.2 recfmt Templates

A recfmt template is a text string that may contain *template spots*. Those spots are substituted in the template using the information of a given record. Any text that is not into a spot is literally copied to the output.

Spots are written surrounded by double curly braces, like:

```
{{...}}
```

Spots contain selection expressions, that are executed every time the template is applied to a record. The spot is then replaced by the string representation of the value returned by the expression.

For example, consider the following template:

```
Task {{Id}}: {{Summary}}
-----
{{Description}}
--
Created at {{CreatedAt}}
```

When applied to the following record:

```
Id: 123
Summary: Fix recfmt.
CreatedAt: 12 December 2010
Description:
+ The recfmt tool shall be fixed, because right
+ now it is leaking 200 megabytes per processed record.
```

The result is:

```
Task 123: Fix recfmt.
-----
The recfmt tool shall be fixed, because right
now it is leaking 200 megabytes per processed record.
--
Created at 12 December 2010
```

Note that you can use any selection expression in the slots, including conditionals and string concatenation.

25.3 recfmt Examples

Apply a template to all the records stored in the 'employees.rec' file:

```
recsel employees.rec | recfmt 'Dear {{Name}}, you are fired.'
```

Read the template from a file:

```
recsel employees.rec | recfmt -f fire-letter.tpl
```

26 csv2rec

csv2rec reads the given comma-separated-values file (or the data in the standard input if no file is specified) and prints out the converted rec data, if possible.

26.1 csv2rec Invocation

csv2rec converts CSV data into rec data. Synopsis:

```
csv2rec [option]... [csv_file]
```

In addition of the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

`'-t type'`

`'--type=type'`

Type of the converted records. If no type is specified then no type is used.

`'-s'`

`'--strict'`

Be strict parsing the csv file.

`'-e'`

`'--omit-empty'`

Omit empty fields.

26.2 csv2rec Examples

Convert from csv to rec:

```
$ csv2rec contacts.csv > contacts.rec
```

Used as a filter, and omitting empty fields:

```
$ cat contacts.csv | csv2rec -e > contacts.rec
```

27 rec2csv

`rec2csv` reads the given `rec` files (or the data in the standard input if no file is specified) and prints out the converted comma-separated-values.

27.1 rec2csv Invocation

`rec2csv` converts `rec` data into CSV data. Synopsis:

```
rec2csv [option]... [rec_file]...
```

The `rec` data can be read from files specified in the command line, or in the standard input. The program echoes the converted data in the standard output.

In addition of the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

```
'-t type'
```

```
'--type=type'
```

Type of the records to convert. If no type is specified then the default records (with no name) are converted.

```
'-S'
```

```
'--sort=field'
```

Sort the output by *field*. This option has precedence to whatever sorting criteria is specified in the corresponding record descriptor with `%sort`.

27.2 rec2csv Conversion

Record sets are not tables, even if tables can be easily emulated using records having the same fields in the same order. For example:

```
a: value
b: value
c: value
```

```
a: value
b: value
c: value
```

```
...
```

There are several ways records are more flexible than tables:

- Fields can appear in a different order in several records.
- There can be several fields with the same name in a single record.
- Records can differ in the number of fields.

Since comma-separated-values files contain tables, the `rec2csv` utility implements an algorithm that deals with the previous difficulties, to generate a table that is what the user expects (likewise).

The algorithm is the following. The utility first scans the specified record set, building a list with the names that will become the table header. For each field, a header is added with the form:

```
FIELDNAME[_N]
```

where `N` is a number in the range `2..inf` and is the “index” of the field in its containing record plus one. For example, consider the following record set:

```
a: a1
b: b11
b: b12
c: c1
```

```
a: a2
b: b2
d: d2
```

The corresponding list of headers being:

```
a b b_2 c a b d
```

Then duplicates are removed:

```
a b b_2 c d
```

The resulting list of headers is then used to build the table in the generated csv file. In this case:

```
"a","b","b_2","c","d"
"a1","b11","b12","c1",
"a2","b2",,"d2"
```

Note how missing fields are implemented as empty rows in the generated csv.

27.3 rec2csv Examples

Generate a csv file out of a rec file:

```
$ rec2csv foo.rec > foo.csv
```

Same operation, but using the program as a filter and specifying a concrete type:

```
$ cat inventory.rec | rec2csv -t Item > items.csv
```

28 mdb2rec

`mdb2rec` reads the given `mdb` file and prints out the converted `rec` data, if possible.

28.1 mdb2rec Invocation

`mdb2rec` converts `mdb` files into `rec` data. Synopsis:

```
mdb2rec [option]... mdb_file [table]
```

All the tables contained in the `mdb` file are exported unless a table is specified in the command line.

In addition of the common options described earlier (see [Chapter 18 \[Common Options\]](#), [page 32](#)) the program accepts the following options.

‘-s’

‘--system-tables’

Include system tables in the output.

‘-l’

‘--list-tables’

Dump a list of the table names contained in the `mdb` file, one per line.

‘-e’

‘--keep-empty-fields’

Don’t prune empty fields in the `rec` output.

28.2 mdb2rec Examples

Access files (*mdb files*) are collections of several relations, also known as tables. Tables can be either *user tables* storing user data, or *system tables* storing information such as forms, queries or the relationships between the tables.

It is possible to get a listing with the names of all tables stored in a `mdb` file by calling `mdb2rec` in the following way:

```
$ mdb2rec -l sales.mdb
Customers
Products
Orders
```

So ‘`sales.mdb`’ stores user information in the tables `Customers`, `Products` and `Orders`. If we want to include system tables in the listing we can use the `-s` command line option:

```
$ mdb2rec -s -l sales.mdb
MSysObjects
MSysACEs
MSysQueries
MSysRelationships
Customers
Products
Orders
```

The tables with names starting with `MSys` are system tables. The data stored in those tables is either not relevant to the `recutils` user (used by the `Access` program to create forms and the like) or is used in an indirect way by `mdb2rec` (such as the information from `MSysRelationships`).

Let’s read some data from the ‘`mdb`’ file. We can get the relation of `Products` in `rec` format:

```
$ mdb2rec sales.mdb Products
%rec: Products
```

```

%type: ProductID int
%type: ProductName size 80
%type: Discontinued bool

ProductID: 1
ProductName: GNU generation T-shirt
Discontinued: 0

...

```

A *record descriptor* is created for the record set containing the generated records, called Products. Note that `mdb2rec` is able to generate type information for the fields. The list of customers is similar:

```

$ mdb2rec sales.mdb Customers
%rec: Customers
%type: CustomerID size 4
%type: CompanyName size 80
%type: ContactName size 60

CustomerID: GSOFTE
CompanyName: GNU Soft
ContactName: Jose E. Marchesi

...

```

The list of orders is a bit more interesting, since it shows how `mdb2rec` manages the relationships between tables in the ‘`mdb`’ file.

```

$ mdb2rec sales.mdb Orders
%rec: Orders
%type: OrderID int
%type: CustomerID size 10
%type: ProductID int
%type: OrderDate date

OrderID: 10248
Customers:CustomerID: FBFOU
Products:ProductID: 1
OrderDate: 2010-08-01T12:30:01

...

```

Both `CustomerID` and `ProductID` are compound fields reflecting references to the record sets Customers and Products.

Note that `mdb2rec` always uses the default role when the name of the column in the referring table is the same than the name of the column in the referred table. This is to avoid redundant compound field names like `Customers:CustomerID:CustomerID:`.

If no table is specified in the invocation to `mdb2rec` all the tables in the file are processed, with the exception of the system tables, for which `-s` shall be used:

```

$ mdb2rec sales.mdb
%rec: Products
...

%rec: Customers

```

...

%rec: Orders

...

Appendix A GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc.

<http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondly, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of “copyleft”, which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The “Document”, below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as “you”. You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A “Modified Version” of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A “Secondary Section” is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The “Invariant Sections” are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The “Cover Texts” are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A “Transparent” copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not “Transparent” is called “Opaque”.

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The “Title Page” means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, “Title Page” means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The “publisher” means any person or entity that distributes copies of the Document to the public.

A section “Entitled XYZ” means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as “Acknowledgements”, “Dedications”, “Endorsements”, or “History”.) To “Preserve the Title” of such a section when you modify the Document means that it remains a section “Entitled XYZ” according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both

covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.
- I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its

Title Page, then add an item describing the Modified Version as stated in the previous sentence.

- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements.”

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See <http://www.gnu.org/copyleft/>.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.3
or any later version published by the Free Software Foundation;
with no Invariant Sections, no Front-Cover Texts, and no Back-Cover
Texts. A copy of the license is included in the section entitled ‘‘GNU
Free Documentation License’’.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with...Texts.” line with this:

```
with the Invariant Sections being list their titles, with
the Front-Cover Texts being list, and with the Back-Cover Texts
being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.